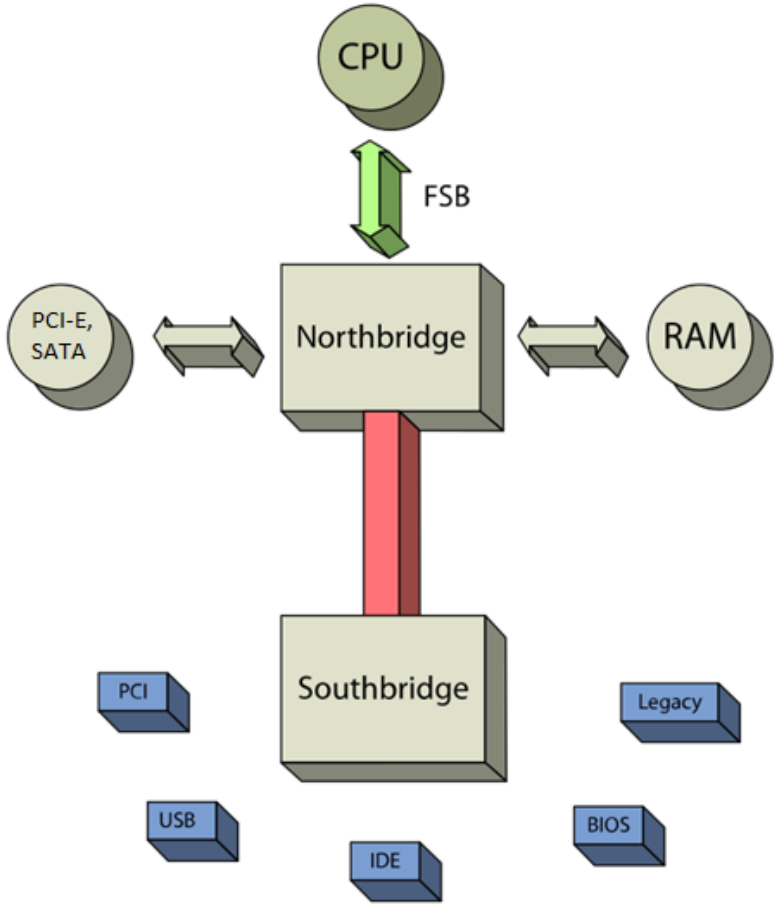
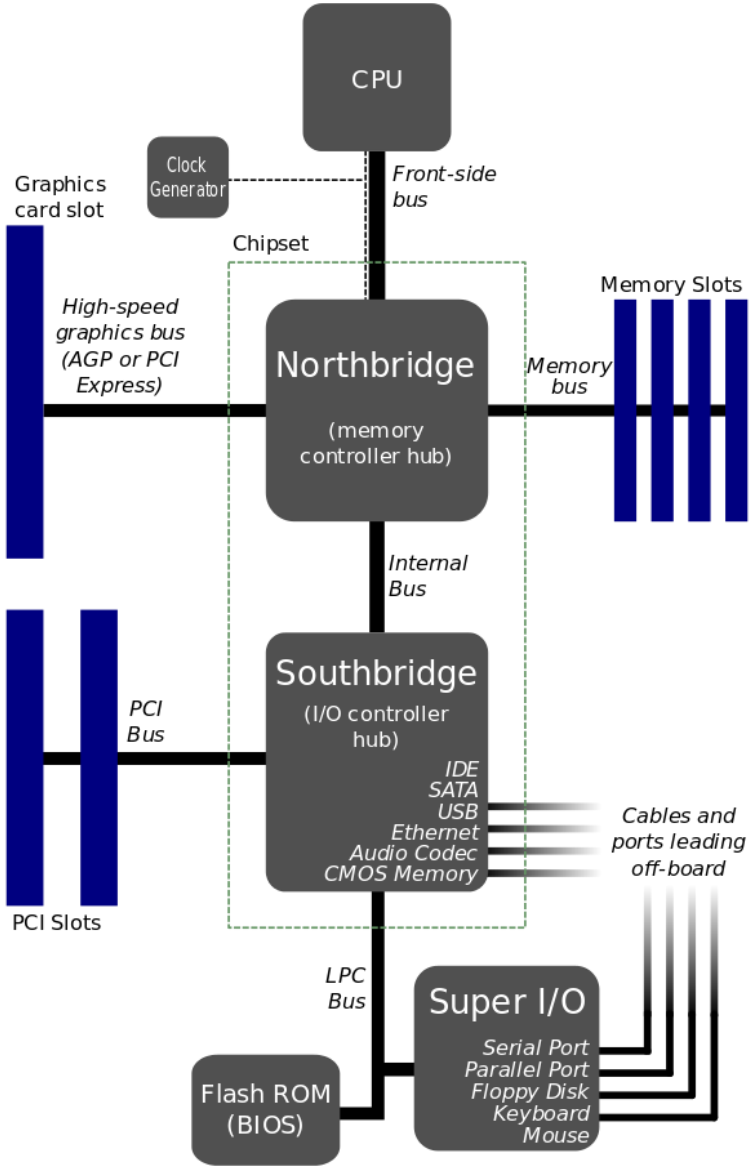
UPG 2

Úvod do jazyka C#

Architektura Počítače





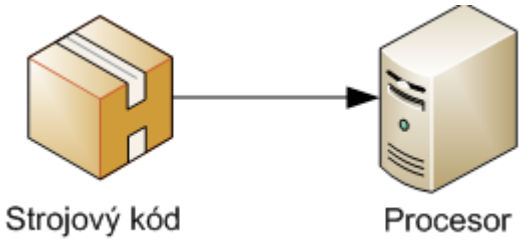
Instrukce CPU

- Jsou to základní příkazy pro provedení operace v CPU
- Posloupnost instrukcí označována jako strojový kód

Vývoj programovacích jazyků
1. generace - Strojový kód

- Elementární (základní, jednoduché) operace
  - sčítání adres, skoky mezi instrukcemi
- Binární kód jedniček a nul => velmi nečitelný => zapisuje se v **šestnáctkové soustavě**
- Ukázka:

```
2104
1105
3106
7001
0053
FFFE
0000
```

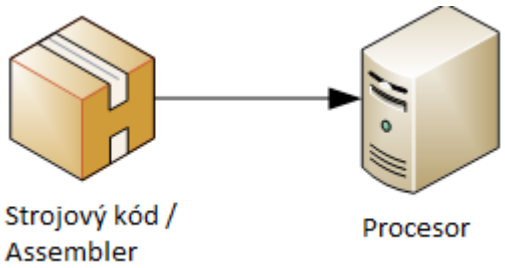


Vývoj programovacích jazyků
2. generace - Assembler

- Stejné jako strojový kód
- Přidány zástupné texty za čísla instrukcí
- Je lidsky čitelnější, programátor si nemusí pamatovat čísla, ale texty

Ukázka:

```
ORG 100
LDA A
ADD B
STA C
HLT
DEC 83
DEC -2
DEC 0
END
```



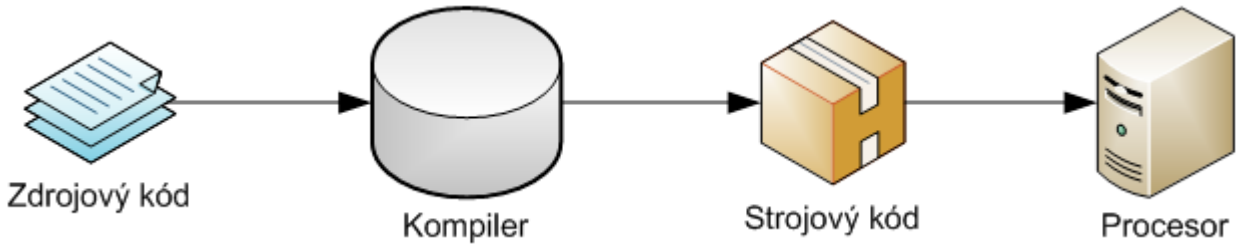
Vývoj programovacích jazyků
3. generace

- Nabízí abstrakci, tj. nejsou závislé na instrukční sadě CPU
- Nezaměřují se na to, jak vidí program počítač, ale jak jej **vidí člověk**
- Výhodou vysoká čitelnost
- Dělí se na 2-*3* kategorie
  - Kompilované jazyky
  - Interpretované jazyky
  - *(Jazyky s virtuálním strojem)*
- Ukázka jazyka 3. generace:
  (součet čísel v jazyce C)

```c
int main(void)
{
        int a, b, c;
        a = 83;
        b = -2;
        c = a + b;
        return 0;
}
```

Vývoj programovacích jazyků
3. generace – Kompilované jazyky

- Kompilované (neřízené) jazyky mají zdrojový kód v jazyce, kterému lidé snadno rozumí.
- Kód se musí **přeložit** do strojového kódu, aby ho bylo možné spustit
- Překlad zajišťuje překladač (**kompiler**), který přeloží celý program najednou do strojového kódu



Vývoj programovacích jazyků
3. generace – Kompilované jazyky

Výhody kompilace

- **Rychlost** – běží stejně rychle, jako by byl program napsán ve strojovém kódu
- **Nepřístupnost zdrojového kódu** – program se šíří jako strojový kód, nelze jej snadno měnit bez původního zdrojového kódu
- **Snadné odhalení chyb ve zdrojovém kódu** – je-li v kódu chyba, proces kompilace selže a programátor dostane chybovou zprávu

Vývoj programovacích jazyků
3. generace – Kompilované jazyky

Nevýhody kompilace

- **Závislost na platformě** – program je závislý na typu CPU, je kompilován do jeho instrukční sady.
- **Nemožnost editace** – jakmile je jednou zkompilován, nelze jej snadno měnit jinak, než novou kompilací *(editovat lze, ale je to práce se strojovým kódem)*
- **Memory management** (Správa paměti) – jelikož počítač programu nerozumí a jen jej mechanicky vykonává, může docházet k chybám s přetečením paměti. Obvykle není správa paměti a jedná se o nižší jazyky. Běhové chyby se kompilací neodhalí.

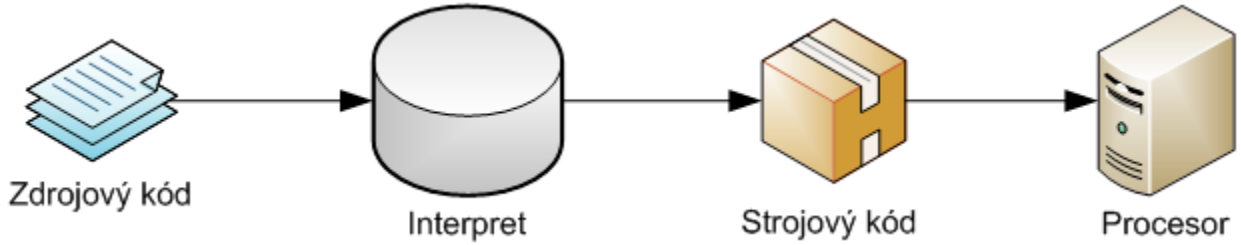
Vývoj programovacích jazyků
3. generace – kompilované jazyky

Příklady jazyků

- C
- C++
- Pascal
- Delphi
- Go

Vývoj programovacích jazyků
3. generace – Interpretované jazyky

- Snaží se **řešit problém přenositelnosti** programů mezi různými platformami (různé typy CPU)
- Místo *kompilátoru* používá **interpreter**
- **Interpreter** znamená tlumočník a stejně jako tlumočník lidských jazyků nepřekládá celý program najednou, ale pouze tu část, která je zrovna potřeba
- **!!! Překlad probíhá za běhu a jen té části, která je zrovna potřeba !!!**



Vývoj programovacích jazyků
3. generace – Interpretované jazyky

Výhody interpretace

- **Přenositelnost** – Program je plně přenositelný, pokud existuje interpret pro danou platformu, půjde tam zdrojový kód programu spustit
- **Jednodušší vývoj** - ve vyšších jazycích jsme odstíněni od správy paměti, kterou za nás dělá tzv. **garbage collector** (často není nutné zadávat datové typy, k dispozici komfortní kolekce a další struktury)
- **Stabilita** – díky tomu, že interpret kódu rozumí, předejde chybám, které by zkompilovaný program jinak klidně vykonal. Běh interpretovaných programů je tedy určitě bezpečnější, dále umožňuje zajímavou vlastnost, tzv. reflexi, kdy program za běhu zkoumá sám sebe
- **Jednoduchá editace** - program lze vyvíjet po částech a nahrávat na cílové umístění, díky tomu, že se nemusí kompilovat, ho je možné jednoduše editovat "za běhu".

Vývoj programovacích jazyků
3. generace – Interpretované jazyky

Nevýhody interpretace

- **Rychlost** - Interpretace může být mnohdy velmi pomalá a program tak plně nevyužívá výkon počítače.
- **Často obtížné hledání chyb** - Díky kompilaci za běhu se chyby v kódu objeví až v tu chvíli, kdy je kód spuštěn. To může být někdy velmi nepříjemné.
- **Zranitelnost** - Protože se program šíří v podobě zdrojového kódu, každý do něj může zasahovat nebo krást jeho části.

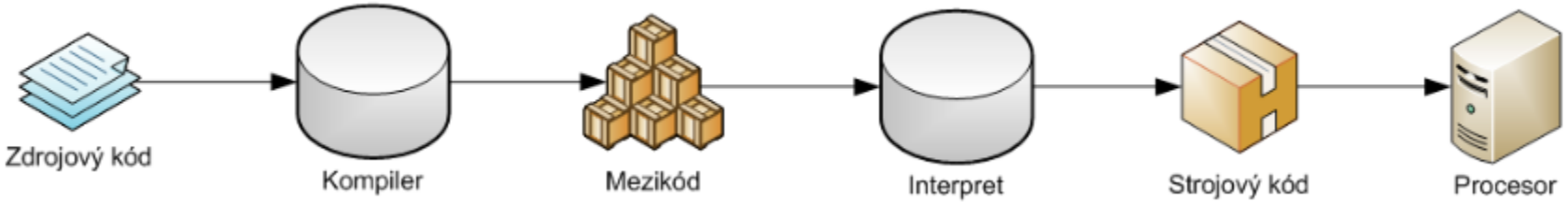
Vývoj programovacích jazyků
3. generace – Interpretované jazyky

Příklady jazyků

- PHP
- Javascript
- Python
- Lisp
- Perl
- PowerShell
- VBA (Excel, …)

Vývoj programovacích jazyků
3. generace – Jazyky s virtuálním strojem

- Kombinace kompilovaných a interpretovaných jazyků
- Zdrojový kód nejprve přeložen do mezikódu **CIL** (Common Intermediate Language), jedná se v podstatě o strojový (binární) kód (**bytecode**). Tento kód má mnohem jednodušší instrukční sadu, než CPU a podporuje OOP => rychle interpretovatelný virtuálním strojem.
- Virtuální stroj – interpreter bytecodu (CIL);
  - Pro .NET je to **CLR** (Common Language Runtime)



Vývoj programovacích jazyků
3. generace – Jazyky s virtuálním strojem

Výhody interpretace

- **Odhalení chyb ve zdrojovém kódu** - Díky kompilaci do CIL se jednoduše odhalí chyby ve zdrojovém kódu.
- **Stabilita** - Díky tomu, že interpret kódu rozumí, zastaví nás před vykonáním nebezpečné operace a na chybu upozorní. Můžeme také provádět reflexi (i když pro CIL, ale od toho jsme většinou odstíněni).
- **Jednoduchý vývoj** - Máme k dispozici hi-tech datové struktury a knihovny, správu paměti provádí garbage collector.

Vývoj programovacích jazyků
3. generace – Jazyky s virtuálním strojem

Výhody interpretace

- **Slušná rychlost** - Rychlost se u virtuálního stroje pohybuje mezi interpretem a kompilerem. Virtuální stroj již výsledky své práce po použití nezahazuje, ale dokáže je cachovat, sám se tedy optimalizuje při četnějších výpočtech a může dosahovat až rychlosti kompileru (Just In time Compilator). Start programu bývá pomalejší, protože stroj překládá společně využívané knihovny.
- **Málo zranitelný kód** - Aplikace se šíří jako zdrojový kód v CIL, není tedy úplně jednoduše lidsky čitelná.
- **Přenositelnost** - Hotový program poběží na každém zařízení, na kterém se nachází virtuální stroj. Nezávislost na samotném jazyce. Na jednom projektu může dělat více lidí, jeden v C#, druhý ve Visual Basic a třetí v C++. Zdrojové kódy se poté vždy přeloží do CIL.

.NET framework

Skládá se ze 4 částí:

1. Jazyk
   - Mnoho programovacích jazyků, např: C#, Java, Visual Basic
2. Visual Studio
   - IDE (Integrated Development Environment), prostředí, ve kterém se píše zdrojový kód a které pomáhá s vývojem
3. Virtuální stroj
   - **CLR** je virtuální stroj, který interpretuje CIL do instrukcí fyzického procesoru.
4. Knihovny
   - Mnoho předpřipravených struktur a komponent např. pro práci s konzolí, databází, formulářovými prvky…

Struktura .NET

- Autorem Microsoft => dobře ladí s MS Windows
- Nutné, aby koncové zařízení obsahovalo správnou verzi .NET
- Každé Windows obsahují nějakou verzi .NET
- Od verze 3.5 podpora jazyka LINQ
- Od verze 4.5 jednoduší psaní asynchronních funkcí